Tree Defect Segmentation using Geometric Features and CNN*

Florian Delconte[1], Phuc Ngo[1], Isabelle Debled-Rennesson[1], Bertrand Kerautret[2], Van-Tho Nguyen[3], and Thiery Constant[4]
florian.delconte@loria.fr

[1] Université de Lorraine, LORIA, ADAGIo, F-54000 Nancy, France
[2] Université Lumière Lyon 2, LIRIS, Imagine, F-69365 Lyon, France
[3] Department of Applied Geomatics, Centre d'applications et de recherche en télédétection, Université de Sherbrooke, 2500 boul.de l'université, J1K2R1 Sherbrooke, QC, Canada
[4] Université de Lorraine, AgroParisTech, INRAE, SILVA, F-54000 Nancy, France

Abstract. Estimating the quality of standing trees or roundwood after felling is a crucial step in forest production trading. The on-going revolution in the forest sector resulting from the use of 3D sensors can also contribute to this step. Among them the terrestrial lidar scanning is a reference descriptive method offering the possibility to segment defects. In this paper, we propose a new reproducible method allowing to automatically segment the defects. It is based on the construction of a *relief map* inspired from a previous strategy and combining with a convolutional neural network to improve the resulting segmentation quality. The proposed method outperforms the previous results and the source code is publicly available with an online demonstration allowing to test the defect detection without any software installation.

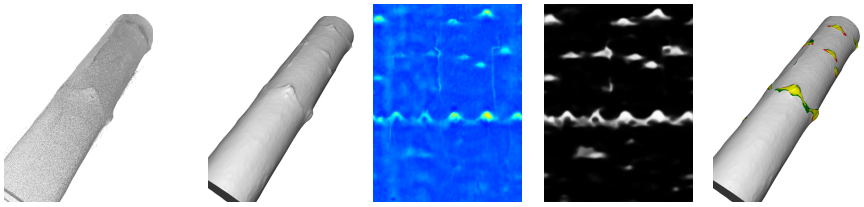
Keywords: Wood surface defects, Defect segmentation, Relief map, LIDAR, Centerline, U-Net

1 Introduction

In the domain of biological image processing, the wood structures are often exploited to address various objectives, for instance, species identification [2], wood quality estimation [7], tree microhabitats identification [21], tracability [23], or plant growing analysis [6]. These various applications rely on different image acquisition modalities such as classical 2D bitmap images (including hyperspectral images), 3D point cloud (from multi-view stereo 3D or LiDAR scan) or 3D volumetric images (medical X-Ray CT scanner [14] or ultra sound [5]).

The aim of this work is to detect defects located on the trunk surface of living tree (see teasing Figure 1). Various types of defects are identified by biology

* This research was made possible by support from the French National Research Agency, in the framework of the project WoodSeer, ANR-19-CE10-011.



(a) Input point cloud (b) Input mesh (c) *Relief map* (d) U-Net result (e) Detection & comparison

Fig. 1: Overview of the proposed method: input LiDAR 3d points (a) and its reconstructed mesh (b) are used to construct the *relief map* (c) which is exploited in U-Net (d). The defects are segmented and compared to ground truth (e).

experts (also called *singularity*) depending on their origins and their development stages (burls, branch scar, picot, ...). Figure 2 illustrates samples of defects on beech and oak species. The detection of such a structure is a key point for the value determination and the optimization of the transformation taking knotiness or aesthetics into consideration. The defect detection on living tree is not an easy task in the image processing domains, since each type of singularities presents numerous geometric variations both inter or intra species. Figure 2 (a,b) illustrates a same defect type on a same species but presenting a very different geometric shape.

(a) Branch scar (beech) (b) Branch scar (beech) (c) Picot (oak)

Fig. 2: Examples of defects: scar and picot. Defect areas are highlighted in red.

In this work, the defect detection is addressed by using 3D scan of trunk (as illustrated in Figure 1). Schütt *et al.* can be considered as the pioneers to exploit the terrestrial LiDAR data for tree defect detection [24]. The authors proposed to localise singularity by combining 3D terrestrial LiDAR with 2D images. After using a cylindrical coordinate transformation, a neural network is trained and used to extract the singularity areas. The method was promising, however the extraction process needs a potential interactive correction and no details are given to reproduce the method nor the result quality measures. Thomas *et al.* proposed later an automatic method to detect severe surface defects by using a

2D circle fitting algorithm [27]. The method was then improved with a parallel implementation in order to reduce execution time [28]. One of the limitation of such an approach, was the minimum detectable defect size was 12.7 centimeters and with a relief higher than 1.27 centimeters. Answering to the previous limitation, Kretschmer *et al.* [13] added more geometry in the singularity detection by using a cylinder fitting based approach. Using a tree reconstruction method [19], a series of cylinders is fitted according to the wood main axis and each 3D points is associated to its cylinder part. Such an association allows to generate a distance map that is used to extract manually the defects. The proposed strategy is not automatic, however it allows to detect smaller defects with near 4.3 centimeters for the minimal size and 2 centimeters for the relief height. Observing that the wood trunk does not always fit perfectly a cylinder, Nguyen *et al.* [18] proposed an automatic patch based method that allows to better follow the trunk geometry. The main algorithm relies on the recovering of the trunk centerline [10] allowing to avoid the cylinder fitting step of previous works.

The methods described in the previous part are designed specifically for the wood defect estimation but it could be interesting to mention other general approaches that are exploited for surface-crack detection in an industrial context. For instance, Tabernik *et al.* [25] proposed a segmentation-based deep-learning method to detect surface anomaly. Even if the considered images differ from the tree defect context, their strategy could be interesting to adapt since the proposed architecture does not request training with numerous images. In other context of the train industry, the defect on rail surface were analysed through [4]. Like the previous works, their approach was based on the deep learning and can detect various defects like weld, squat or joint. Finally, we can refer to another application of micro cup surface inspection from a confocal laser microscopy images that exploits neural network to detect defects on very textured images [30].

Following the previous strategy introduced by Nguyen *et al.* [18], we propose a new method based on the construction of a new *relief map* image combined with a convolutional neural network (CNN) to precisely segment tree defects. The main overview of the method is given on the Figure 3. The *relief map* is constructed from the input mesh (upper part of Figure 3) and the convolutional neural network is exploited to segment defect area (lower part of Figure 3) that can be visualized on the original input mesh.The first step of the new approach is described in the following section (Section 2) with the overview of the centerline detection, followed by *relief map* image construction. Associated to this representation, the segmentation process based on the U-Net architecture is introduced in Section 3. The experiment part presenting the main results and reproducibility links are addressed in Section 4 before concluding.

2 Geometric tools

The proposed method relies on three geometric tools: (i) the *centerline of the wood log*, (ii) the *reference and delta distances* and (iii) the *relief map*. The first

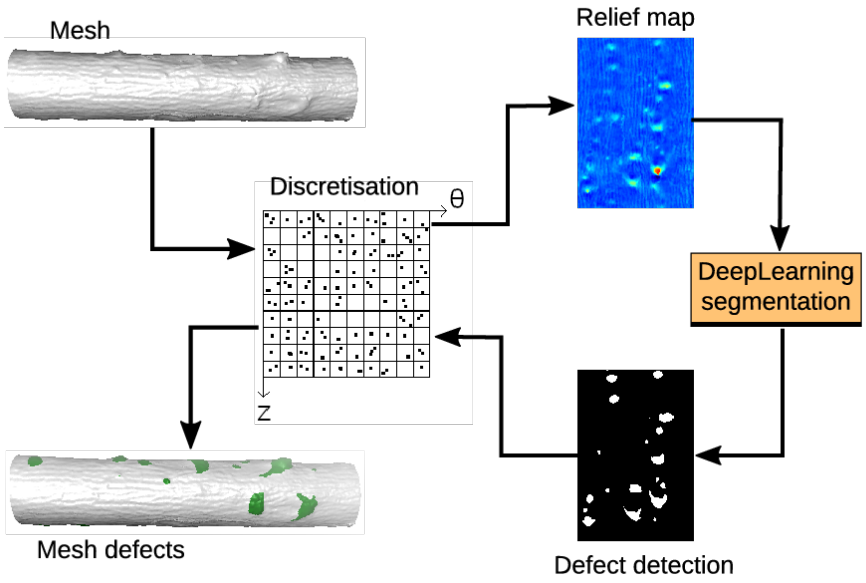


Fig. 3: Pipeline of the proposed method.

two tools are introduced in [10,18] and summarized in following sections. The third tool, the *relief map*, is defined in Section 2.3. It permits to represent the input 3D points with a 2D map characterising the relief of the points, relatively to the centerline of the wood log and from a fitted tangent plane.

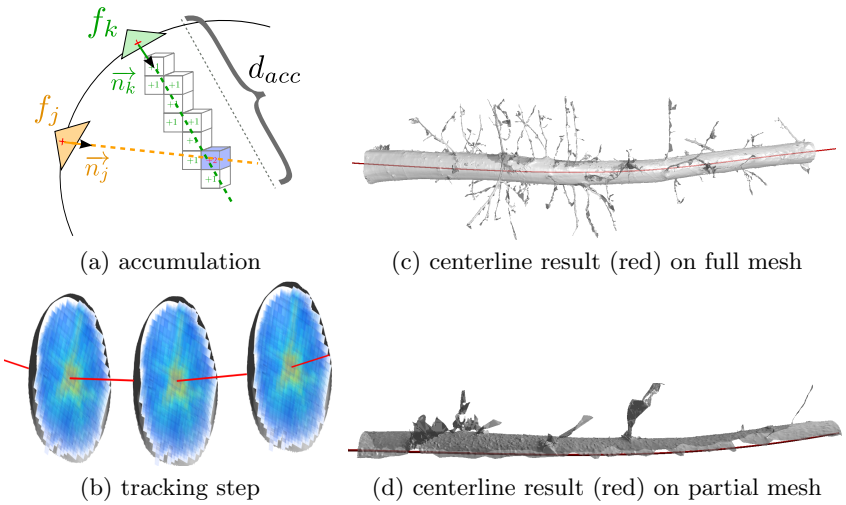
2.1 Centerline of the wood log

In [10] a method is presented to extract the centerline of 3D shapes using solely partial mesh scans of the shapes. The centerline is a polyline with several small segments (see Figure 4 (b,c)). It is obtained by constructing an accumulation map from input faces and normal vectors (see Figure 4 (a)) and by filtering it with a confidence vote. Since the method inputs are only a set of faces, the centerline can also be recovered both from full and partial mesh (see Figure 4 (b,c)). The details of the algorithm are available in the associated reference [11] and on the *GitHub* repository:

`https://github.com/kerautret/CDCVAM`

Due to the non constant diameter of wood logs, a process of optimization must be done to obtain a smooth centerline of the wood logs. In [18], the authors used a smoothing process based on cubic spline. Note that the implementation details and reproductive evaluation can be found in the complementary work [17] with the *GitHub* repository:

`https://github.com/vanthonguyen/treelogdefectsegmentation`



(a) accumulation

(c) centerline result (red) on full mesh

(b) tracking step

(d) centerline result (red) on partial mesh

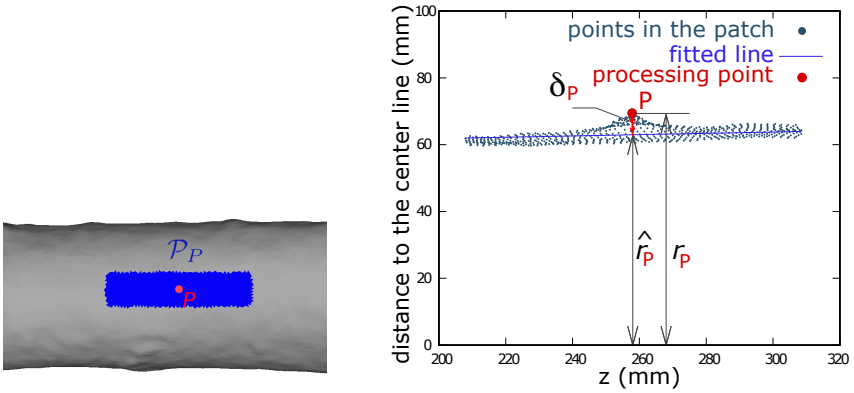
Fig. 4: Illustration of the main idea of the centerline extraction algorithm: (a) accumulation step from surface faces f_k and f_j in the direction of their normal vectors ($\overrightarrow{n_j}$ and $\overrightarrow{n_k}$); (b) example of tracking step from the 3D accumulation values. Images (c) and (d) show the centerline extraction respectively on full and partial mesh.

2.2 Reference and delta distances

In order to easily access the neighborhood of each point on the wood log surface, we work in cylindrical coordinates. A local coordinate system (C_i, u_i, v_i, w_i) is defined for each segment S_i of the centerline. A point $P(x, y, z)$ in Cartesian coordinates corresponds to the cylindrical coordinates (r_P, θ_P, z_P) with:

- r_P is the distance between P and P', the projection of P on the segment S_i of the centerline.
- z_P is the height of P along the centerline.
- θ_P is the angle formed between the segment PP' and the axis v_i of local coordinate system associated to S_i.

For more details of transformation in cylindrical coordinate, we refer the readers to [18]. To correctly detect the local relief variation around each point P of the wood log, a rectangular neighborhood is studied, named *patch* $\mathcal{P}_P$, it is proportional to the size and circumference of the wood log (see Figure 5 (a) and [18] for details). $\mathcal{P}_P$ characterizes the shape of the log around the point P. The central straight line fitting the points of $\mathcal{P}_P$ is calculated by a RANSAC based linear regression. The ***reference distance of the point P***, $\hat{r}_P$, is the distance from this straight line to the centerline (see Figure 5 (b)). The difference between r_P and $\hat{r}_P$ represents the relief of the tree at a point. It is called the



(a) Rectangular patch $\mathcal{P}_P$ centered on $\mathcal{P}$ (b) Delta distance

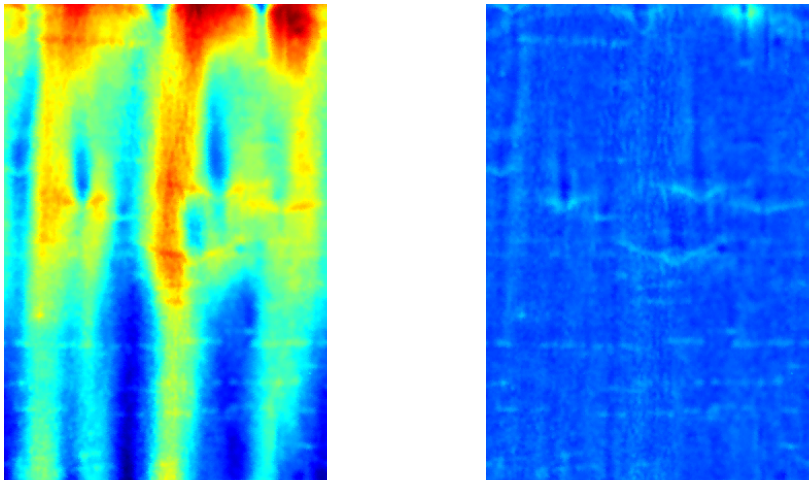
Fig. 5: (a) A patch in blue, associated to the red point, is used to compute the reference distance of this point. (b) Computation of the reference distance for the red point. See [18] for more details.

delta distance: $\delta_P = r_P - \hat{r}_P$ (see Figure 5 (b)). We use the delta distance in the next section to generate the relief maps.

2.3 Relief map

A *relief map* is a 2D representation of the tree mesh. It is obtained by firstly discretizing the cylindrical point space and by secondly, completing the missing information with a multi-resolution analysis. This map is used to segment the defects. It must also allow a reverse operation, *i.e.,* compute from pixels of the map the corresponding 3D points of the mesh.

Cylindrical space discretisation The relief map represents the *unfolding of the wood log*. The width of the map is the circumference of the trunk, *i.e.,* $2\pi * r_m$ with r_m the average radius of the trunk. The height of the map is the height of the trunk, obtained by subtracting the z component of the point having the maximum height and the one of minimum height. Each point of the tree mesh is associated to a cell of the relief map. We then calculate a value to represent all the points of a cell. The chosen value is the **maximum value of the delta distances** of the points associated to the cell. An illustration is given in Figure 6. The two maps are generated from the same input mesh. On the left, intensity of the pixels is calculated from the distance to the centerline. On the right, intensity of the pixels is calculated from the delta distances. With the map obtained by the distance to the centerline, we can observe the artefacts, the yellow and red traces, due to the non-cylindrical tree, while using the delta distance, these traces disappear and the defects become more visible. Figure 7, on the first line, provides several relief maps deduced from this process.



(a) Distances to the centerline (b) Relief map

Fig. 6: Difference between the distance map (a) and the relief map (b).

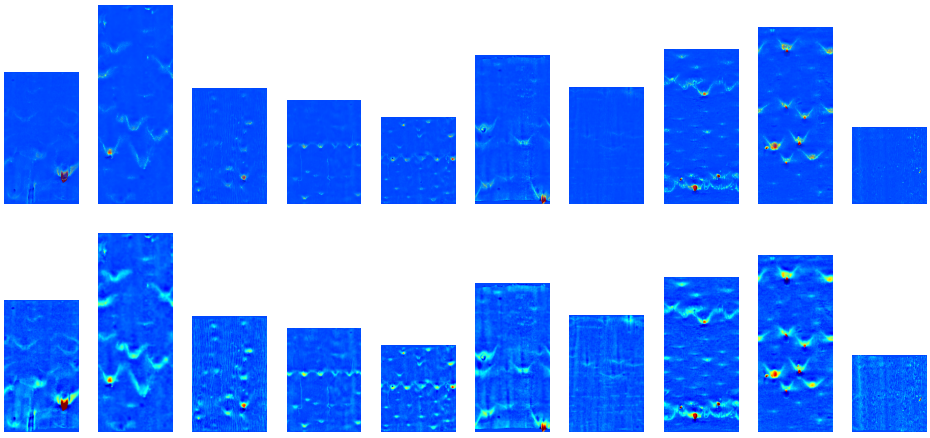


Fig. 7: Examples of relief maps. Red color for the stronger reliefs. Blue color for the lower reliefs. The upper row is not improved with a multi-resolution analysis.

Multi-resolution analysis It is possible that some cells of the relief map do not contain a value because no point is associated with it. To handle this, we propose a multi-resolution analysis to improve the obtained relief map. In the proposed process, for every empty cell, we reduce the resolutions by a factor $\frac{1}{2^n}$, with $n \in \mathbb{N}$, until the cell contains at least one point. It should be noticed that if n is too high, we may lost information of delta distance for the defect detection. Therefore, during the multi-resolution analysis, we fix the limite of n to four. Figure 8 illustrates the multi-resolution analysis to obtain a value in an empty

cell. We consider in this example an array T with a resolution of 10×10. Some cells contain black dots, corresponding to the points previously discretized. In others, there is none, like the cell located in $(2, 2)$. The multi-resolution analysis is illustrated by the colors of the borders of the table. Respectively, the black, red and blue borders correspond to resolutions reduced by a factor of $\frac{1}{1}$, $\frac{1}{2^1}$, $\frac{1}{2^2}$. We look for the information in the cell $(2, 2)$, then $(1, 1)$ of the red discretization, then $(0, 0)$ of the blue discretization, etc ... The value in the $(2, 2)$ cell is then the maximum of the delta distances of the points in the $(0, 0)$ cell of the blue discretization. To represent the discretization in the form of an image we associate a gray intensity to the delta distance. This intensity is distributed on a fixed scale: 1cm equals ten gray levels starting from -5. The colored relief maps in this article are obtained by applying a color scheme from blue to red. The bottom row in Figure 7 shows the improvement brought to the relief maps with the multi-resolution analysis. We can see in Figure 9, results of the multi-resolution analysis centered on a branch scar type defect.

3 Segmentation with U-Net architecture

Hereafter, we process the detection of defects on tree barks using the previously obtained relief map. Note that the 3D problem of defect detection on tree bark surfaces becomes a 2D problem of relief map segmentation. More precisely, it is a binary-image-classification in which each pixel of the relief map will be classified as defect or not. We can observe in Figure 7, that the defects on tree barks may have arbitrary size, shape and orientation. Furthermore, the roughness of the tree bark, the variability of the defects on the same species and between the different species make the detection task difficult to automate by conventional segmentation algorithms.

Over the past few years, the deep-learning methods are becoming common and successful for the segmentation task with remarkable performance improvements. Indeed, they often achieve the highest accuracy rates on popular segmentation benchmarks comparing to the classical computer-vision approaches. Furthermore, the deep-learning algorithms can be adapted to different problems as they can learn the hidden high-level features from the image directly, and have capacity to represent and recognize the complex structures.

In this paper, we use a deep learning-based segmentation method, namely U-Net [22], to detect tree bark defects with the relief maps as input. It should be mentioned that in the context of surface-singularity detection, several neural network architectures have been proposed (see Section 1). As stated in [22], U-Net enables the model to be trained using a small number of samples, and to create a precise pixel-wise mask of interest objects in the images. Thus, it is a well-suited architecture for our segmentation problem. In the following, we describe a modified version of the original U-Net [22] for detecting tree bark defects, and the generation of training data from the relief maps.

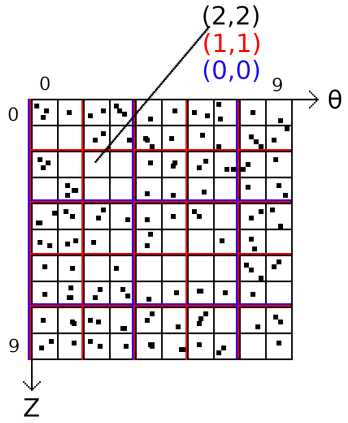


Fig. 8: Illustration of the multi-resolution analysis on an example of size 10×10. Ajouter description

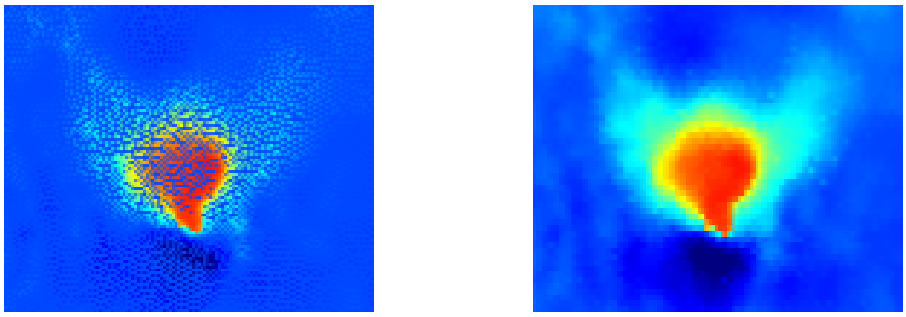


Fig. 9: Effect of multi-resolution analysis. Left: Without multi-resolution processing, we can see the missing pixels. Right: A relief map completed with multi-resolution analysis.

3.1 Segmentation network

The U-Net was first introduced in [22] as a fully convolutional network (FCN) architecture for biomedical image segmentation, and it was designed for a precise pixel-wise segmentation. Recently, many variants of U-Net architecture have been proposed to address the segmentation of medical images, satellite images such as U-Net++ [31], KU-Net [29], TernausNet [8], ... U-Net is well-known for its performance to be trained with very few training images.

U-Net is an auto-encoder architecture. The encoder stage takes the input images and extracts features from objects in the image, then condenses them into smaller layers. These features are propagated in the decoder stage to produce a segmentation.

The original U-Net proposes to use convolution layers followed by max pooling for down-sampling in encoding part, while the decoding part consists of

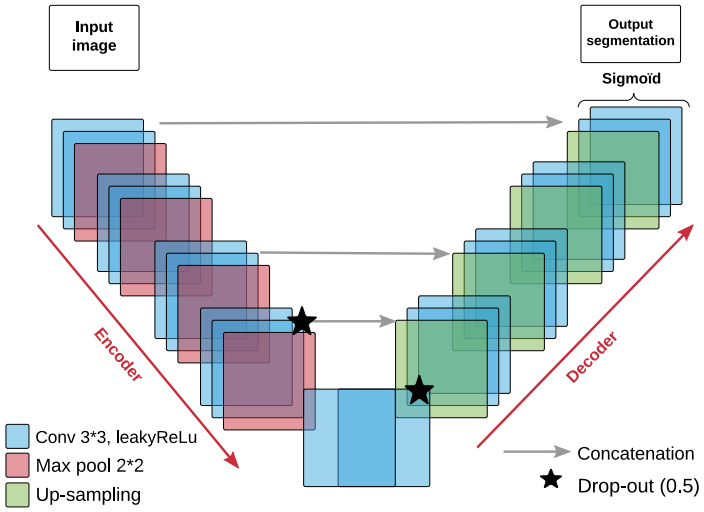


Fig. 10: The architecture for tree bark defect detection based on U-Net [22].

up-sampling followed by concatenation with the corresponding layer of the encoding part. Each layer is followed by the rectify linear unit (ReLu) activation function, except the last one. This final layer is a 1x1 convolution followed by a pixel-wise soft-max over the final feature map. The cross entropy loss function is used to update weights of the network. The soft-max function redistributes the weights of the final layer of the network in the interval of $[0, 1]$ modeling a probability distribution over predicted output classes. In total the network has 23 convolutional layers. More details of U-Net can be found in [22].

In this paper, to address our problem of detecting tree bark defects which is defined as two class segmentation, we made several changes to the original U-Net. In order to reduce the over-fitting of the considered neural network, we apply a regularization technique, called *drop-out*. More precisely, we add two dropout layers in the encoder and decoder, with probability 0.5, to randomly drop some of the connections between layers. In addition, due to the dying ReLU problem [16] –*i.e.,* the ReLU neurons become inactive and only output 0 for any input– the Leaky ReLu activation function is employed instead of ReLU from the original architecture. Finally, in the last layer, we use a Sigmoid activation function instead of soft-max function to ensure the output pixel values range between 0 and 1. For the training, we use input image of size 320x320 pixels. The proposed network architecture is illustrated in Figure 10.

3.2 Training data

The training dataset for U-Net framework is built from the relief maps generated from tree bark surfaces by the process described in Section 2. It should

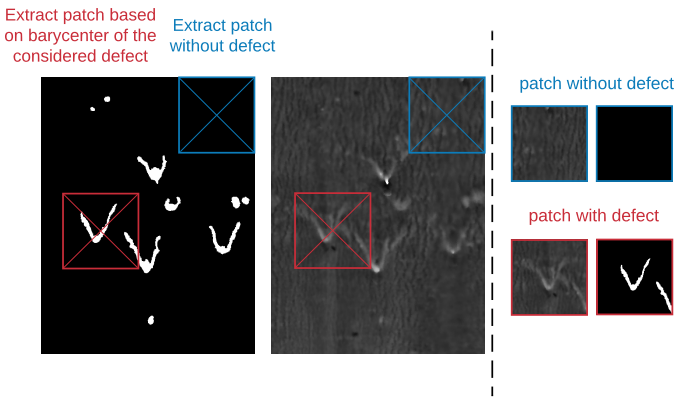
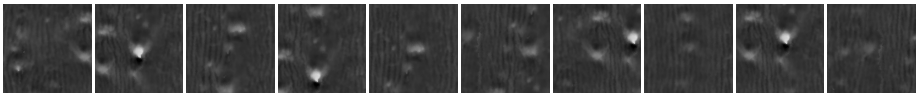


Fig. 11: Illustration of extracting patches of size 320x320 pixels from the relief and the annotated maps.

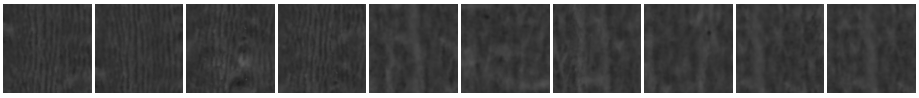
be mentioned that we only have 25 annotated meshes –*i.e.,* 25 relief maps with ground-truths– of tree bark surfaces for the learning process (for more details, see Section 4.1). Due to this limited number of samples, two strategies have been adopted to augment the existing data while keeping the significant characteristics of learning objects which are tree bark defects. First and foremost, we split each relief map into patches of same size. For this, we carry out two types of cutting (see Figure 11). The first one aims to obtain samples centered on defects. More precisely, we perform the splitting of relief map according to the barycenter of the connected component associated to the defect in the annotated image. Note that if a defect is close to the border, a translation is applied to obtain a patch containing the defect and being included in the map. The second one collects samples that do not contain any defect so that the network can learn tree bark without defect. Some samples of extracted patches with and without defects are given in Figure 12. It should be mentioned that the size of the generated patches is limited by the width and height of our relief maps. As observed in Figure 7, the relief maps may have different sizes because of the discretization being made with respect to the circumference and height of the tree bark (see Section 2). For our framework, the patches are of size 320x320 pixels which is the largest size that could be extracted from the relief maps. The obtained images are then randomly separated into two subsets with a ratio of 7:3 for the training and validation of the network.

After this splitting process, different transformation techniques have also been used on the obtained patches for data augmentation. In particular, we consider the operations: rotation, vertical and horizontal flip, zoom and deletion of rectangular area randomly [3]. Note that this data augmentation is performed on the fly, *i.e.,* during the training time.



(a) Extracted patches with defects from relief maps

(b) Extracted patches with defects from annotated maps



(c) Extracted patches without defect from relief maps

Fig. 12: Some samples from training data.

4 Experiments

4.1 Dataset

We have two datasets for our experiments: **INRAE1a** and **INRAE1b**. INRAE1a contains 10 trunks of different species: beech (1), birch (1), elm (1), fir (2), red oak (2), wild cherry (2) and service wood (1). INRAE1b contains 15 meshes including alder (4), aspen (4), beech (1), birch (2), horn beam (1), lime (1), red oak (2). The first dataset was used in [18], experiments were carried out on INRAE1a to compare performance and robustness of our method with [13] and [18] on different tree species. The relief maps of INRAE1a are given in Figure 14. The second dataset is used for the training and illustrated in Figure 13.

Both datasets have the ground-truths being made by hand-labeling defects. The ground-truth is given as a set of point indices associated to the defect. These indices are then used to generate the annotated maps for training the network.

4.2 Network training

The training process was first performed on the relief maps generated from 15 meshes of INRAE1b. We used the parameters recommended in [17] for computing these maps. After splitting the obtained relief maps into patches (see Section 3.2), we divide the patches of each map into two subsets: 70% for training and 30% for validation. This subdivision allows to have the same bark variability on training and validation. To summarize, the whole dataset has 265 images of size 320x320 pixels, it is partitioned into 204 images for training and 61 images for validation.

The training process is done on GPU (geforce RTX 2080Ti with 12Go RAM). The modified U-Net is implemented using *Tensorflow 2.2* [26] and *Keras* [9]. During the training, data augmentation was applied randomly to the input images,

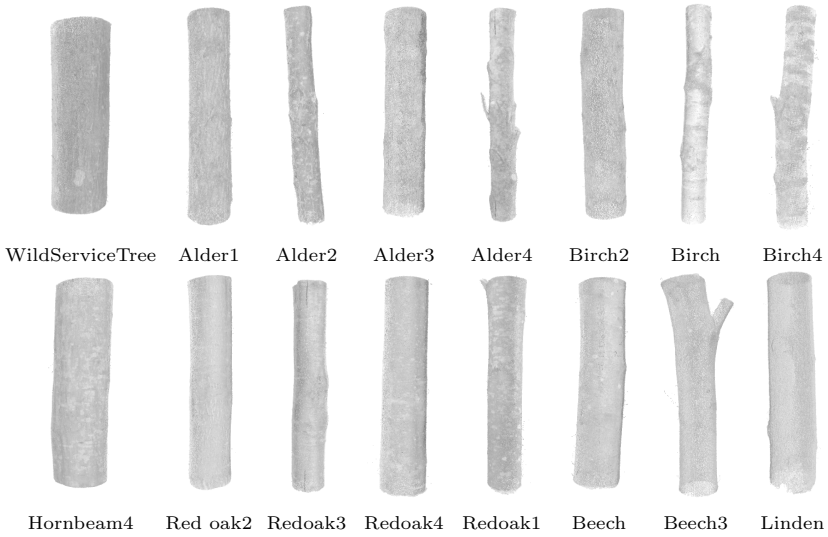


Fig. 13: Illustration of input 3D points of the mixed test.

including rotation, vertical and horizontal flip, zoom and deletion of rectangular area randomly [3]. We used the Adam optimiser [12], and set the learning rate at 0.0001, the two parameters $\beta_1 = 0.9$ and $\beta_2 = 0.99$ (default values in *Tensorflow 2.2*). We trained our network for 40 epochs, each epoch comprised 63 steps with 10 images per batch. About the parameters of dropout rate δ and Leaky ReLu activation α, several values have been tested, and we come out with $\delta = 0.5$ and $\alpha = 0.01$ for the smallest loss function (binary cross entropy) on the validation. It should be mentioned that our training is quite fast, it takes about 14 seconds per epoch. In other words, the proposed architecture allows a high-quality segmentation and very fast training –about **10 minutes** for the whole training process– with very small dataset. In particular, the prediction takes, on average, 451 miliseconds per map.

4.3 Experimental results

The first experiments were performed on INRAE1a. More precisely, the relief maps were generated for the 10 meshes of the dataset, then predicted by our network which is previously trained on INRAE1b. The output prediction is a gray-level image. We threshold this image at 0.5 to obtain a binary image in which the white pixels indicate the tree defect and black is not. The results are given in Figure 14.

To evaluate and compare the methods, we used the classic metrics: precision, recall and F measure (F1). For a fair comparison, we performed the evaluation measures on the mesh points, as done in [18], but not the predicted maps. As

described in Section 2.3, from pixel positions, we can easily retrieve the mesh point indices associated to the pixel, and identify those points for localizing the defects on the mesh.

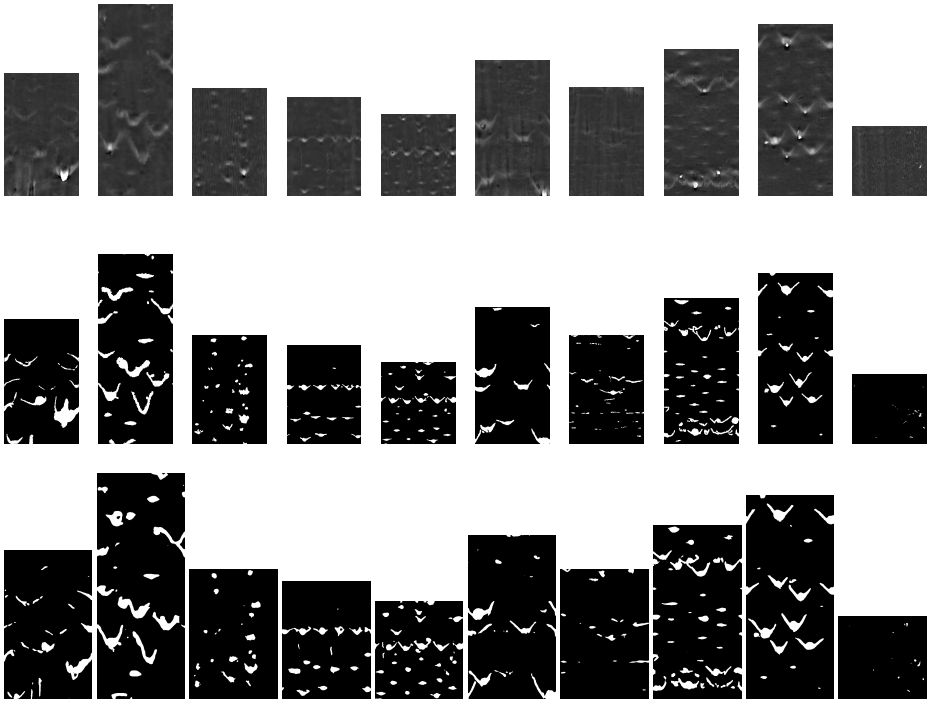


Fig. 14: Results on INRAE1a. First row: relief maps, second row: ground-truths, Third row: predictions by our network.

Table 1 shows the obtained results. Generally, the proposed method outperforms both cylindrical-based [13] and patch-based [18] methods. We improve the detection performance, about 41% and 8% better in F1 measure comparing to [13] and [18], respectively. Figure 15 shows a visual comparison on meshes of the results obtained by the proposed method and the patch-based method [18]. We can see in Figure 15 (a) that our method tends to produce fewer false positives, but sometimes miss small defects. This may due to the fact that the network has not been trained with samples containing small defects. Figure 15 (b) is an example where the detection by our method covers better the form of defects than patch-based method [18].

We carried out a second experiment to demonstrate the generalization of the network. Using the same parameters, we trained our CNN on mixed data of INRAE1a and INRAE1b, i.e., 25 meshes in total. We generated 5 different folds, each of which contains 20 meshes for training and 5 for validation. Table 2

INRAE1a	Patch method [18]			Cylinder method [13]			Our method		
	prec	recall	F1	prec	recall	F1	prec	recall	F1
Fir1	**0.747**	0.769	0.757	0.137	**0.937**	0.238	0.746	0.857	**0.797**
Fir2	0.673	0.775	0.719	0.353	0.452	0.395	**0.792**	**0.801**	**0.795**
WildCherry1	0.696	0.765	0.728	0.683	0.512	0.584	**0.757**	**0.881**	**0.813**
WildCherry2	**0.846**	0.711	0.771	0.661	0.822	0.732	0.799	**0.955**	**0.870**
Redoak1	0.749	**0.742**	0.744	0.479	0.444	0.459	**0.866**	0.696	**0.770**
Redoak2	0.428	**0.833**	**0.564**	0.061	0.400	0.104	**0.730**	0.428	0.538
Beech	0.670	**0.604**	0.634	0.360	0.289	0.320	**0.863**	0.591	**0.701**
Birch	0.733	**0.756**	0.744	0.607	0.421	0.496	**0.774**	0.726	**0.748**
Elm	0.694	**0.755**	0.721	0.494	0.309	0.378	**0.881**	0.642	**0.741**
WildServiceTree	0.247	**0.741**	0.370	0.057	0.463	0.100	**0.856**	0.504	**0.633**
Overall	0.685	0.740	0.710	0.289	0.563	0.380	**0.793**	**0.789**	**0.790**

Table 1: Comparison results: Overall row is computed from the sum of TP,TN,FP and FN on all the tested meshes.

summarizes the distribution of data and the results obtained. For each fold in Table 2, the meshes indicated in *Mesh id* correspond to test data, and the others are used for training. In this way, we ensure to test and compare our method with the others on all available data. It can be observed that, over the 5 folds, we generally obtain the best F1 measure, and almost better on the precision comparing to [13] and [18]. The worst scoring result by the proposed method is *Beech3* in the *fold 5*, we are at 0.336 for F1. Though, this score is comparable to the best score of 0.493 obtained by [18].

Note that the measured results of the methods [18] and [13] for the dataset INRAE1a (see Table 2) are from [18], while the results on the new dataset in Table 2 are obtained by using the source code from the *GitHub* repository with the recommended parameters described in [17,18].

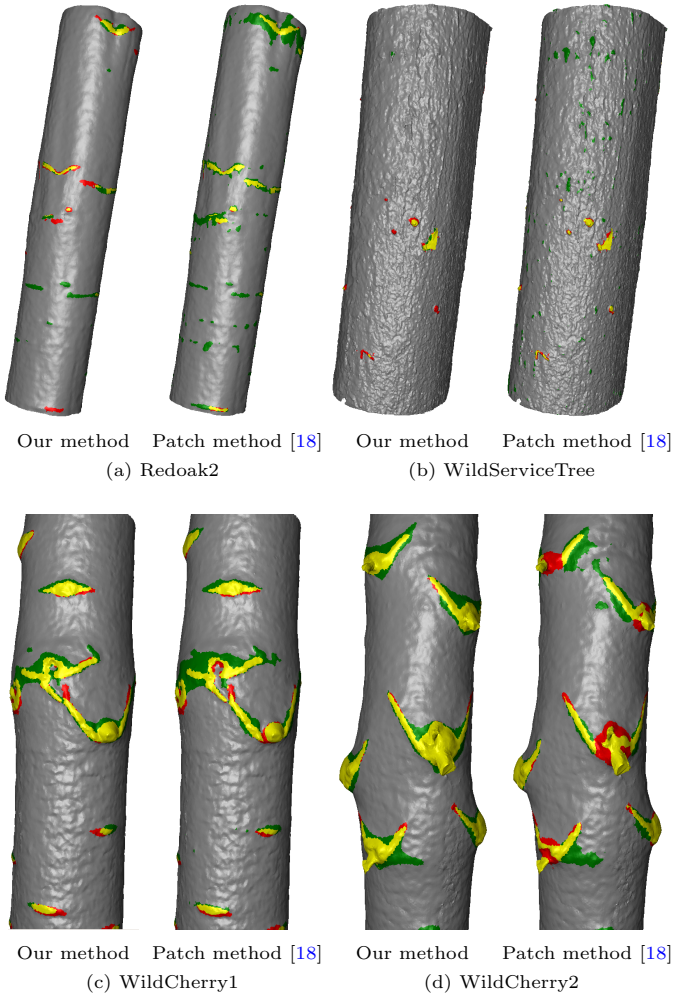
An online demonstration for testing the proposed method is available at:
https://kerautret.github.io/TLDDC/

5 Source code to reproduce results

5.1 Global view

The source code to reproduce the results presented in the article, including *relief map* and segmentation, is available at the *GitHub* repository:
https://github.com/FlorianDelconte/TLDDC
The repository is composed of different file and directories :

- The directory **Centerline** contains the centerline code.
- The directory **examples** contains INRAE1e and INRAE1b meshes, each mesh is accompanied by two files indicating the ground-truth location of



(a) Redoak2 (b) WildServiceTree

(c) WildCherry1 (d) WildCherry2

Fig. 15: Comparison on mesh. Yellow is true positive, red is false negative and green is false positive compared to the ground-truth.

defects using mesh faces and mesh points, suffixed by *-groundtruth.id* and *-groundtruth-points.id* respectively.

- The directory **models** contains the trained models in the paper: five files of *.hdf5* extension for the corresponding k-fold, and one named *KFoldAssociation* for a relation file between mesh example and the k-fold.
- The directories **mesures** and **run** contain python and bash scripts to directly reproduce the results of this article.
- The code for generating the relief maps is found in *UnrolledMap.h*, *UnrolledMap.cpp*, *DefectSegmentationUnroll.h* and *DefectSegmentationUnroll.cpp*.

Fold number	Mesh id	Patch method [18]			Cylinder method [13]			Our method		
		prec	recall	F1	prec	recall	F1	prec	recall	F1
1	Beech	**0.872**	0.511	0.643	0.325	0.316	0.318	0.853	**0.655**	**0.740**
	Birch	0.647	**0.843**	0.731	0.577	0.445	0.500	**0.818**	0.692	**0.749**
	Alder1	0.089	**0.746**	0.158	0.044	0.380	0.075	**0.798**	0.391	**0.524**
	Aspen1	0.462	0.721	0.562	0.174	0.671	0.274	**0.831**	**0.794**	**0.811**
	Hornbeam4	0.308	0.802	0.445	0.016	**0.954**	0.030	**0.779**	0.589	**0.669**
Overall		0.480	**0.687**	0.563	0.089	0.484	0.150	**0.831**	**0.687**	**0.750**
2	Elm	0.771	0.669	**0.715**	0.158	**0.670**	0.253	**0.889**	0.578	0.699
	Fir1	0.600	**0.824**	0.693	0.107	0.758	0.187	**0.831**	0.819	**0.824**
	Alder2	0.446	0.796	0.571	0.283	0.530	0.366	**0.612**	**0.843**	**0.707**
	Aspen2	0.666	**0.619**	**0.641**	0.352	0.303	0.323	**0.917**	0.419	0.574
	Birch4	0.673	0.572	0.616	0.509	0.532	0.518	**0.694**	**0.819**	**0.750**
Overall		0.618	**0.674**	0.643	0.199	0.527	0.286	**0.759**	0.673	**0.712**
3	Fir2	0.656	0.814	0.725	0.279	0.514	0.360	**0.820**	**0.822**	**0.820**
	Redoak1	0.706	**0.743**	0.723	0.487	0.373	0.420	**0.834**	0.681	**0.748**
	Alder3	0.414	**0.556**	0.474	0.352	0.431	0.385	**0.775**	0.381	**0.510**
	Aspen3	0.544	0.523	0.532	0.116	0.386	0.175	**0.892**	**0.579**	**0.701**
	Linden	0.791	0.600	0.681	0.154	**0.950**	0.264	**0.874**	0.644	**0.740**
Overall		0.624	**0.648**	0.635	0.189	0.565	0.281	**0.845**	0.635	**0.724**
4	Redoak2	0.428	**0.827**	**0.562**	0.062	0.474	0.108	**0.802**	0.350	0.486
	WildCherry1	0.680	0.772	0.721	0.679	0.593	0.632	**0.826**	**0.805**	**0.814**
	Alder4	0.901	0.625	0.737	0.782	0.489	0.601	**0.952**	**0.756**	**0.841**
	Aspen4	0.897	0.454	0.602	0.341	**0.729**	0.463	**0.953**	0.632	**0.759**
	Redoak4	0.479	**0.760**	**0.587**	0.086	0.271	0.12	**0.815**	0.431	0.563
Overall		0.749	0.617	0.676	0.379	0.589	0.460	**0.904**	**0.699**	**0.787**
5	WildCherry2	0.788	0.744	0.765	0.807	0.674	0.733	**0.852**	**0.943**	**0.894**
	WildServiceTree	0.262	**0.732**	0.384	0.051	0.479	0.090	**0.856**	0.559	**0.675**
	Beech3	0.497	**0.491**	**0.493**	0.143	0.192	0.16	**0.867**	0.231	0.364
	Birch2	0.395	**0.595**	0.474	0.083	0.175	0.108	**0.804**	0.454	**0.580**
	Redoak3	0.563	0.618	0.587	0.143	**0.656**	0.232	**0.862**	0.622	**0.722**
Overall		0.564	**0.638**	0.597	0.222	0.480	0.301	**0.851**	0.627	**0.721**

Table 2: Comparison results on mixed dataset. Overall row is computed from the sum of TP,TN,FP and FN on all the tested meshes.

5.2 Installation

For compilation process, the program required this libraries to be installed:

- DGtal 1_1_0 or later : https://github.com/DGtal-team/DGtal
- Eigen3 : https://eigen.tuxfamily.org/dox/GettingStarted.html
- GNU GSL : https://www.gnu.org/software/gsl/
- PCL : https://pointclouds.org/downloads/

To use the segmentation models, these following dependencies are necessary:

- Tensorflow2.2 : https://www.tensorflow.org/install/pip
- tensorflow-addons : https://www.tensorflow.org/addons/overview

– openCV : https://pypi.org/project/opencv-python/

Instructions for installing on ubuntu 20.04 and debian 10 have been tested and are detailed on *GitHub*. Once the dependencies are installed and the sources downloaded, the code is built using the following commands:

```
cd TLDDC
mkdir build
cd build
cmake .. -DDGtal_DIR=/path/to/DGtalSourceBuild
make
```

Two executable files are generated in the **build** directory:

– *segunroll* allows to generate relief maps from meshes.
– *segToMesh* allows to project the segmentation of the defects of the relief map towards the mesh.

5.3 Usage

To generate the relief map, run the following command from **build**:

```
./segunroll -i InputMesh [-h] [-n] [CenterlineParameters] [ReliefMapParameters]
```

With

– *InputMesh* is the path to a trunk mesh.
– *-h* is the option for the command line helper.
– *-n* is the option allows to invert the normals of the faces of the meshes[5].
– *CenterlineParameters* contains the parameters of centerline computation (*--accRadius*, *--trackStep*, *--binWidth*, *--patchWidth*, *--patchHeight*, *--voxelSize*). They are set by default with the recommended values in [17].
– *ReliefMapParameters* contains the parameters for the relief map (*--decreaseFactor*, *--grayscaleOrigin*, *--intensityPerCm*). They are also set with default values.

The following files are created after executing the command:

– *centerline.off*: the generated centerline.
– *discretisation.txt*: the discretization map.
– *output.pgm*: the generated relief map.

To segment the bark tree defects, run the following command from **build**:

```
python3 ../run/predict.py InputReliefMap PathToModel Threshold
```

With

– *InputReliefMap* contains the path to the relief map.

[5] The normals must be directed towards the interior of the tree

- *PathToModel* contains the model file, one of the five in the **models** directory.
- *Treshold* contains the threshold ($[0; 255]$) to apply on the network prediction.

The following files are created after executing the command:

- *outputSEG.pgm*: the prediction result by the network.
- *outputSEGTRESH.pgm*: the segmented image after thresholding.

To project the segmented result on the input mesh, run the following command from **build**:

```
./segToMesh -i InputMesh
```

With *InputMesh* contains the path to the same mesh used to generate the relief map. The following files are created after executing the command:

- *output-defect.id*: the file containing the id of the points of the mesh belonging to a defect (to compare with groundTruth).
- *Poutputdefect.off*: the output mesh with the segmented defects in green.

To execute these three scripts in succession, run the command from **run**:

```
./deep-segmentation.sh PathToModel InputMesh Treshold
```

To reproduce the measurements in Table 1, run the following command from **mesures**:

```
./testINRAE1A.sh
```

This command fill the *results.tex* file which contains the performance measure presented in this article.
To reproduce the Table 2 table, run the follwing command from **mesures**:

```
./testK_folds.sh
```

After executing this command, five files are created: *resultsN.tex* (with $N = 1...5$) containing the performance measure corresponding to the lines in Table 2.

6 Conclusion

From the difficult problem of tree defect detection, new contributions is proposed in this work with first a new relief map able to locally adapt itself on global shape of the trunk. Such an adaption is important since the trunk geometry may appear with significant variations which make wrong the segmentation of the defect. The second contribution, is to propose a segmentation process based on the U-Net architecture allowing to outperform the previous works. The results, source code and dataset are all available on a git repository allowing the reproduction of the results together with an online demonstration.

In future works, we plan to address the defect classification. Such features will be interesting to get a finer estimation of the wood quality. Other perspectives consist in investigating the 3D point cloud semantic segmentation such as PointNet [20], ConvPoint [1], PointCNN [15], ...

References

1. Boulch, A.: Convpoint: Continuous convolutions for point cloud processing. Computers & Graphics (2020)
2. Carpentier, M., Giguère, P., Gaudreault, J.: Tree species identification from bark images using convolutional neural networks. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1075–1081 (2018)
3. Devries, T., Taylor, G.W.: Improved regularization of convolutional neural networks with cutout. CoRR **abs/1708.04552** (2017)
4. Faghih-Roohi, S., Hajizadeh, S., Núñez, A., Babuska, R., De Schutter, B.: Deep convolutional neural networks for detection of rail surface defects. In: International Joint Conference on Neural Networks (IJCNN). pp. 2584–2589 (2016)
5. Gilbert, G.S., Ballesteros, J.O., Barrios-Rodriguez, C.A., Bonadies, E.F.*et al.*.: Use of Sonic Tomography to Detect and Quantify Wood Decay in Living Trees. Applications in Plant Sciences **4**(12) (Dec 2016)
6. Gélard, W., Herbulot, A., Devy, M., Casadebaig, P.: 3D leaf tracking for plant growth monitoring. In: 2018 25th IEEE International Conference on Image Processing (ICIP). pp. 3663–3667. IEEE (2018)
7. He, T., Liu, Y., Yu, Y., Zhao, Q., Hu, Z.: Application of deep convolutional neural network on feature extraction and detection of wood defects. Measurement **152**, 107357 (2020), publisher: Elsevier
8. Iglovikov, V., Shvets, A.: Ternausnet: U-net with VGG11 encoder pre-trained on imagenet for image segmentation. CoRR **abs/1801.05746** (2018)
9. Keras: https://keras.io/guides/functional_api/
10. Kerautret, B., Krähenbühl, A., Debled-Rennesson, I., Lachaud, J.O.: Centerline detection on partial mesh scans by confidence vote in accumulation map. In: 2016 23rd International Conference on Pattern Recognition (ICPR). pp. 1376–1381 (Dec 2016)
11. Kerautret, B., Krähenbühl, A., Debled-Rennesson, I., Lachaud, J.O.: On the implementation of centerline extraction based on confidence vote in accumulation map. In: International Workshop on Reproducible Research in Pattern Recognition. pp. 116–130. Springer (2016)
12. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (12 2014)
13. Kretschmer, U., Kirchner, N., Morhart, C., Spiecker, H.: A new approach to assessing tree stem quality characteristics using terrestrial laser scans. Silva Fennica **47** (01 2013)
14. Krähenbühl, A., Kerautret, B., Debled-Rennesson, I., Mothe, F., Longuetaud, F.: Knot segmentation in 3D CT images of wet wood. Pattern Recognition **47**(12), 3852–3869 (Dec 2014)
15. Li, Y., Bu, R., Sun, M., Wu, W., Di, X., Chen, B.: Pointcnn: Convolution on x-transformed points. In: Advances in neural information processing systems. pp. 820–830 (2018)
16. Lu, L., Shin, Y., Su, Y., Karniadakis, G.E.: Dying relu and initialization: Theory and numerical examples (2019)
17. Nguyen, V.T., Kerautret, B., Debled-Rennesson, I., Colin, F., Piboule, A., Constant, T.: Algorithms and implementation for segmenting tree log surface defects. In: International Workshop on Reproducible Research in Pattern Recognition. pp. 150–166. Springer (2016)

18. Nguyen, V.T., Kerautret, B., Debled-Rennesson, I., Colin, F., Piboule, A., Constant, T.: Segmentation of defects on log surface from terrestrial lidar data. In: 2016 23rd International Conference on Pattern Recognition (ICPR). pp. 3168–3173. IEEE (2016)
19. Pfeifer, N., Gorte, B., Winterhalder, D.: Automatic reconstruction of single trees from terrestrial laser scanner data. Int. Arch. of Photogram. Remote Sensing & Spatial Information Sciences **35** (01 2004)
20. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 652–660 (2017)
21. Rehush, N., Abegg, M., Waser, L.T., Brändli, U.B.: Identifying tree-related microhabitats in TLS point clouds using machine learning. Remote Sensing **10**(11), 1735 (2018), publisher: Multidisciplinary Digital Publishing Institute
22. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. CoRR **abs/1505.04597** (2015)
23. Schraml, R., Entacher, K., Petutschnigg, A., Young, T., Uhl, A.: Matching score models for hyperspectral range analysis to improve wood log traceability by fingerprint methods. Mathematics **8**(7), 1071 (2020)
24. Schütt, C., Aschoff, T., Winterhalder, D., Thies, M., Kretschmer, U., Spiecker, H.: Approaches for recognition of wood quality of standing trees based on terrestrial laserscanner data. Laser-scanners for forest and landscape assessment. Int. Archives of Photogrammetry, Remote Sensing, and Spatial Information Sciences **36**, 179–182 (2004)
25. Tabernik, D., Sela, S., Skvarc, J., Skocaj, D.: Segmentation-based deep-learning approach for surface-defect detection. Journal of Intelligent Manufacturing pp. 1–18 (2020)
26. Tensorflow: https://www.tensorflow.org/api_docs
27. Thomas, L., Shaffer, C.A., Mili, L., Thomas, E.: Automated detection of severe surface defects on barked hardwood logs. Forest (10171) (2006)
28. Thomas, R.E., Thomas, L.: Using parallel computing methods to improve log surface defect detection methods. In: 18th International Nondestructive Testing and Evaluation of Wood Symposium; 2013 September 24-27; Madison, WI. Gen. Tech. Rep. FPL-226. Madison, WI: US Department of Agriculture, Forest Service, Forest Products Laboratory: 196-205. pp. 196–205 (2013)
29. Wagner, F., Ipia, A., Tarabalka, Y., Lotte, R., Ferreira, M., P.M, A., Gloor, M., Phillips, O., Aragão, L.: Using the u-net convolutional network to map forest types and disturbance in the atlantic rainforest with very high resolution images. Remote Sensing in Ecology and Conservation **5** (03 2019)
30. Weimer, D., Thamer, H., Scholz-Reiter, B.: Learning defect classifiers for textured surfaces using neural networks and statistical feature representations. Procedia CIRP **7**, 347–352 (12 2013)
31. Zhou, Z., Siddiquee, M.M.R., Tajbakhsh, N., Liang, J.: Unet++: A nested u-net architecture for medical image segmentation (2018)